MICRO PHOTON DEVICES

QRN

Quantum Random Number Generator

© Micro Photon Devices S.r.l.
Via Stradivari, 4
39100 Bolzano (BZ), Italy
email info@micro-photon-devices.com
Phone +39 0471 051212 • Fax +39 0471 501524

QRN User Manual Version 3.1.0 - November 2013

Contents

Theory of operation

Summary

The Quantum Random Number Generator (QRNG), designed by Micro Photon Devices (MPD) shown in Figure 1 and named QRN, is a physical generator of random numbers based on the intrinsic randomness of quantum optical processes. The device exploits the stochastic number of detected photons within a fix observation time, in order to generate a random bit stream. The detection and processing of the photons is performed by a CMOS chip which includes 1024 independent Single Photon Avalanche Diodes (SPAD) [1] and achieves random bit generation rates of 16, 32, 64, 128 and 200 Mbit/s, depending on the device grade. MPD has verified the absolute randomness of the generated random data with standard statistical test suites and more stringent correlation and bias tests. In all cases, the data generated by the QRN passed the applied tests.

Introduction

Different applications require the generation of high quality random number sequences, such as Cryptography, Monte Carlo numerical simulations and image processing [2] [3] [4] [5], just to mention a few. Cryptography, for example, aims to keep messages safe. A private message, sent via physical media in form of electromagnetic waves or electrical signals, must be read only by the recipient it is addressed to. The degree of security of the message depends on the cipher and the key used for encoding.

Figure 1: MPD Quantum Random Number Generator

In the optimal case, the cipher must guarantee a broad space in which the key is selected, and the probability to choose a specific key must be uniformly distributed between all possible values. The latter requirement is often achieved by selecting the key using randomly generated numbers. Ideally, a random number generator provides unbiased and unpredictable data. Accordingly, the output of the generator is completely independent from the sequence of previously generated numbers.

Three major families of Random Number Generators (RNG) exist: Pseudo-RNG (PRNG), Chaotic RNG (CRNG) and Quantum RNG (QRNG). Pseudo-RNGs produce sequences which look like random but are, in fact, generated by deterministic algorithms. Although PRNG are very fast and cost-effective, they require a seed sequence to initialize the generator state. The outcomes are then completely predictable, periodical and seed-dependent.

CRNG are instead based on chaotic physical processes, complex systems in which a small variation of the initial conditions produces large changes of some observables. Typically, the CRNGs use thermal noise [6], optical fluctuations of laser radiation [7], jitter of oscillators in integrated circuits [8], just to cite a few. High quality random streams are obtained, but the generators themself are not intrinsically random. In addition, an attacker could get access to the chaotic system and simultaneously measure the same physical observables and reproduce the random data.

Conversely, Quantum RNGs are based on truly quantum physical processes, whose randomness is guaranteed by theory and experiments. In addition, an attacker, who would attempt to measure the physical observables which are used to generate the data, would destructively perturb the system, in most cases, or would not be able to clone the intercepted message, e.g. to replicate the number of photons measured by a photodetector.

A special class of QRNG is the one based on optical phenomena such as the reflection or transmission of photons by a semi-transparent mirror or the number of measured photons within a fix sampling time. MPD's QRN falls in this category since, as said, exploits the stochastic number of detected photons within a fix observation time. The system is based on a monolithic CMOS chip containing an array of independent cells, each capable of detecting single photons and properly generating random bits. The photon detection is performed by a SPAD in each cell. The array-based architecture of the chip allows to boost the rate of processed photons, hence random bits, per second. With 1024 cells, laid out as an array of 32 columns and rows, the present chip achieves a maximum bit rate of 200 Mbit/s. This fast bit-rate makes the chip, for example, suited for "One-Time Pad" quantum cryptography systems, in which a random bit is used for each information bit sent through the communication channel [8].

Structure of the generator

As already mentioned above, the photon detector exploited by the QRN is a SPAD [17] [18]. A SPAD is a diode that is reverse-biased well above its breakdown voltage. Under this operating condition, a single event of quantum nature, namely the absorption of a single photon, causes the generation of an electron-hole pair, which is accelerated by the high electric field across the junction. The energy of the free carriers is sufficient to trigger a macroscopic avalanche current of few milliamperes through the device. Proper electronics senses and signals the event. The use of a SPAD as a detector has several advantages. Firstly, an event can be triggered both by the "internal" thermal generation process and the "external" photons. A second advantage is that no analogue measurement of voltage or current is needed, since the detector acts as a digital Geiger counter. It follows that no read-out noise is added to the measurement process and no electrical noise affects the ignition of the events, hence the randomness. The QRN is based on an array of 1024 independent SPAD detectors, organized as a 32x32 matrix, fabricated in standard CMOS technology such as the one described in [9].

As shown in Figure 2, each SPAD is controlled by a front-end electronics [10], which provides the correct bias voltage above breakdown, senses the avalanche current and quenches it. After ignition, the SPAD is then reset back to operation after a so-called dead-time. A custom 8-bit Linear-Feedback Shift-Register (LFSR) counts the events and provides them to the off-chip electronics. The LFSR can process a large number of quantum triggering events (e.g. hundreds) before producing the corresponding output bits.

A FPGA provides the control signals to the chip. It sets the read-out rate of the bits and reads the counter values. In addition, it implements a basic Pseudo-Random Number Generator (PRNG), KISS (Keep It Simple Stupid) [11], which is used to enhance the security level of the QRN. This generator is initialized once during the start-up phase of the device by using a quantum random stream and it runs indefinitely. When the security option is activated, the XOR between the QRN stream and the PRNG is performed. The output stream remains random even in case of a failure of the QRN. It's worth noting that the PRNG and the security option were never active during the validation tests of the MPD's QRN.

As shown in Figure 2, the random stream is then processed inside the FPGA by a bit-selector, which removes some of the bits from the stream according with their position in the LFSR counter. The bit-selector is adjusted depending on the average number of quantum events detected by each cell of the array, in order to obtain an equal probability of obtaining zero and ones. Variable bit-rates from 16 Mbit/s to 200 Mbit/s of random data are obtained after this processing depending on the device grade.

Finally, the stream is sent to a FIFO register for buffering the random bits and maximizing the transfer rate to the user PC. By choice, the FIFO data can be directed to a serial hardware output (HW-OUT), which provides 3.3 V LVCMOS logic values for the random bits.

In order to provide a high quantum event generation rate, the chip (i.e. the SPAD detectors) is illuminated by means of a light emitting diode, driven by a current generator controlled via a digital filter.

The filter monitors the total number of detection events during several read-out operations and adjusts the LED current to keep the mean detection rate constant. In this way, the QRN always operates under optimal conditions, independently of variations of physical parameters like LED efficiency, array sensitivity, warm up at power on, aging, etc. This controller is not critical for the random bit generation process, but it is necessary to assure the performance and the stability of the device over long operating times. In case of failure of the illumination source, the system stops and generates an error message to the user. Since the typical dead-time of the SPAD front-end is 60 ns, a maximum number of $16.6*10^6$ photons per second and per SPAD can be acquired. It follows that up to $17*10^9$ events per second are available to generate the random stream with the MPD's QRN.

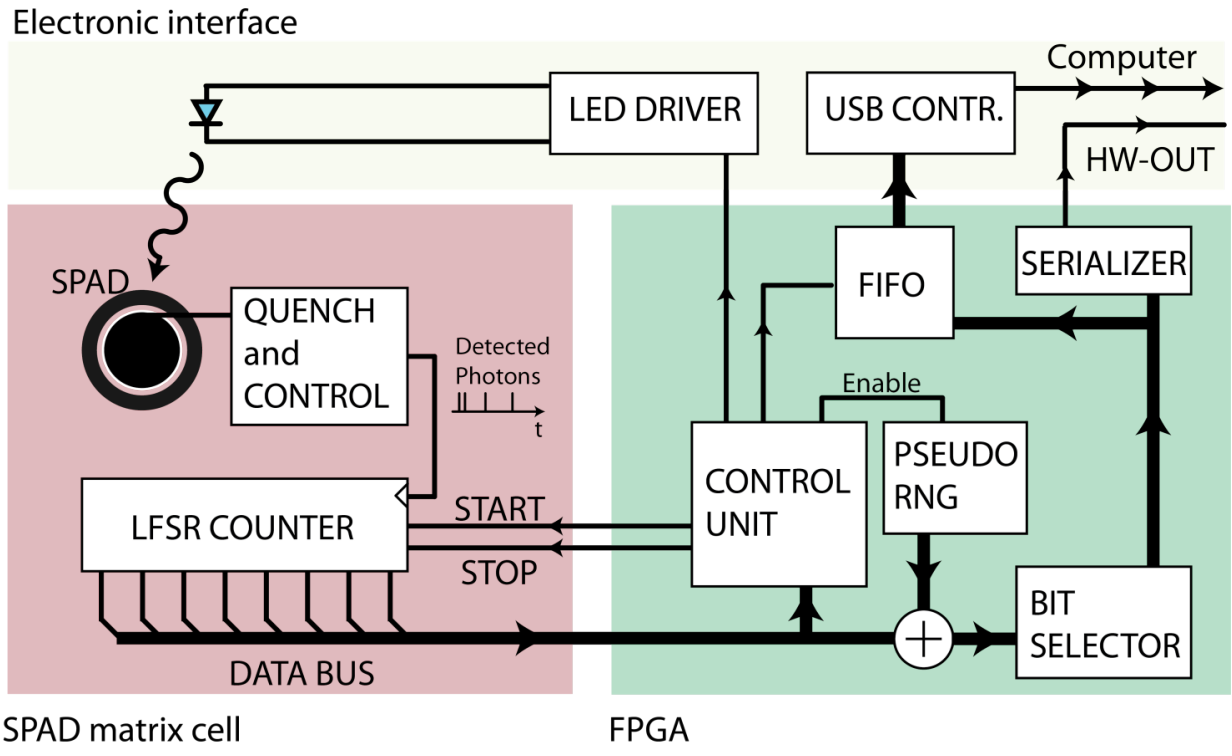


Figure 2. Scheme of a single cell of the 32x32 CMOS array. Each independent cell contains a SPAD detector, a front-end electronics and a Linear Feedback Shift Register (LFSR) for counting the quantum events. A FPGA controls the read-out of the chip and the USB interface to transmit the random binary stream to the computer.

Embedded security test

The QRN performs a continuous test of the random bit stream before providing the data to the user. A failure of the generator or an active attack to the system is immediately recognized and the generator enters an idle state. It is sufficient to reset the device to bring it back to the normal operating state. A simple frequency test is embedded in the device electronics, which looks for extremely rare events as the generation of 48 consecutive '0' or '1'. One of these events might occur, for example, once every 7 years for a device which generates data 24^h per day at a rate of 64Mbit/s. Although possible, this event is extremely improbable.

Qualification of the QRNG

In order to test the quality of the random binary stream generated by the QRNG, two of the most popular test suites for random number generators, *Dieharder* [12] and *TestU01* [13], were modified to support the QRN. The C libraries, which are used to control the generator, were directly integrated in the source code of both test suites. The reason was to prove the long-term stability of the QRN in generating random numbers at high rates and at frequent request of new data streams. The original test suites were also applied to random data stream from the QRN, previously stored on the hard-drive. No differences on the results of the tests were observed.

The first applied test suite has been *Dieharder. Dieharder* is an extension of the "Diehard Battery of Tests of Randomness" [15], a very well-known random number test suite developed by G. Marsaglia. In particular, *Dieharder* was extended with specific tests to check for bit-level randomness of sequences produced by physical RNG (Monobit, Runs and Serial). Furthermore, it incorporates tests from the Statistical Test Suite (STS) developed by the National Institute for Standards and Technology (NIST) [14]. Table 1 shows the results of 10 complete and independent executions of the *Dieharder* test suite. In most cases, the obtained p-values are above 0.01 and below 0.99 and only few exceptions are obtained outside these boundaries, producing a warning message. However, the number of suspect p-values is within the expected statistical failure rate, assuming a 1% significance level. In addition, the p-values are well uniformly distributed within 0 and 1. The mean values, calculated over the single test types and executions, are sufficiently close to 0.5. The only exception to this is the *diehard_sums* test that seems to have some unsolved bugs, and it is thus considered not reliable by the developer himself (see [16]). As a comparison, cryptographically secure random number generators, as the Advanced Encryption Scheme (AES), and the XOR between the QRN and KISS PRNG pass the test with similar p-value distributions.

Table 1. Summary of the results of ten executions of the statistical test suite Dieharder. The p-values are listed for each test family. (*) Several tests are repeated more than once in the test suite with different parameters. Thus, only the average p-value is reported. (**) diehard_sums generates p-values which are not uniformly distributed. Similar p-value distributions were obtained by executing the test suite with AES and making the XOR between the QRNG output and the KISS PRNG (data not shown). The results suggest that diehard_sums is not a reliable test, as already stated in the Dieharder documentation [12].

Test	1	2	3	4	5	6	7	8	9	10	Mean
diehard_birthdays	0,3076	0,3885	0,8915	0,0920	0,2119	0,0421	0,6225	0,6221	0,6560	0,9465	0,4781
diehard_operm5	0,6092	0,7151	0,9232	0,3270	0,3980	0,4632	0,9970	0,9085	0,5261	0,4593	0,6327
diehard_rank_32x32	0,7586	0,0148	0,3911	0,2398	0,0673	0,4608	0,4535	0,9969	0,3683	0,2394	0,3991
diehard_rank_6x8	0,2750	0,4677	0,3459	0,9769	0,4058	0,9870	0,5612	0,8559	0,8352	0,1099	0,5821
diehard_bitstream	0,0664	0,6368	0,1847	0,8392	0,3445	0,3888	0,3830	0,3433	0,9891	0,4336	0,4609
diehard_opso	0,4493	0,9904	0,2566	0,7402	0,8743	0,4962	0,3230	0,9607	0,9712	0,0681	0,6130
diehard_oqso	0,1313	0,6521	0,9631	0,9070	0,9346	0,5601	0,5631	0,0604	0,7495	0,8364	0,6358
diehard_dna	0,1121	0,4518	0,3289	0,1370	0,0001	0,3805	0,1516	0,7131	0,3625	0,9875	0,3625
diehard_count_1s_str	0,9560	0,7377	0,2618	0,8614	0,2909	0,4300	0,0870	0,9520	0,2478	0,9515	0,5776
diehard_count_1s_byt	0,2358	0,7591	0,7140	0,1464	0,4822	0,4741	0,1418	0,4085	0,0355	0,0617	0,3459
diehard_parking_lot	0,8727	0,1671	0,9570	0,5313	0,6291	0,9717	0,3695	0,3190	0,7734	0,2540	0,5845
diehard_2dsphere	0,7778	0,8088	0,4922	0,9094	0,8781	0,1495	0,7450	0,7547	0,9576	0,8871	0,7360
diehard_3dsphere	0,6588	0,9591	0,5813	0,9995	0,9444	0,1866	0,2974	0,4312	0,8888	0,4421	0,6389
diehard_squeeze	0,5869	0,4116	0,7330	0,7908	0,6881	0,2039	0,7038	0,2007	0,2033	0,7529	0,5275
diehard_sums **	0,0097	0,0717	0,0359	0,2323	0,0091	0,0053	0,3168	0,1191	0,6860	0,0201	0,1506
marsaglia_tsang_gcd	0,5372	0,9716	0,5264	0,1508	0,1409	0,0160	0,4270	0,4694	0,9310	0,7774	0,4948
marsaglia_tsang_gcd	0,8496	0,8879	0,1440	0,8815	0,3175	0,9092	0,2900	0,0506	0,3872	0,9979	0,5715
sts_monobit	0,0232	0,9088	0,8883	0,8851	0,8387	0,3365	0,4675	0,8459	0,9966	0,5923	0,6783
sts_runs	0,6791	0,7988	0,6423	0,8294	0,7319	0,1705	0,2974	0,6352	0,9967	0,7853	0,6567
rgb_kstest_test	0,9513	0,8005	0,5268	0,2059	0,6548	0,6139	0,5529	0,1210	0,4686	0,3972	0,5293
dab_bytedistrib	0,5512	0,0672	0,3609	0,6737	0,5163	0,3615	0,3049	0,4640	0,3203	0,0973	0,3717
dab_dct	0,1387	0,3679	0,5654	0,1441	0,1231	0,0406	0,4712	0,2342	0,8333	0,9845	0,3903
dab_monobit2	0,0425	0,2548	0,1360	0,9773	0,6560	0,1601	0,7439	0,5859	0,3931	0,6717	0,4621
sts_serial*	0,4584	0,4807	0,5967	0,5733	0,6404	0,5538	0,6001	0,5897	0,6433	0,5137	0,5650
rgb_bitdist*	0,6155	0,3641	0,5680	0,5508	0,5550	0,5925	0,6804	0,6519	0,5825	0,4730	0,5634
rgb_minimum_distance*	0,1913	0,2326	0,4021	0,4785	0,3705	0,3996	0,6690	0,5090	0,7495	0,4966	0,4499
rgb_permutations*	0,5470	0,6697	0,4188	0,5457	0,3954	0,4412	0,5461	0,8310	0,6045	0,6972	0,5697
rgb_lagged_sum*	0,4910	0,5497	0,6195	0,5459	0,5702	0,5379	0,4864	0,4917	0,5671	0,6088	0,5468
dab_filltree*	0,2873	0,8643	0,8436	0,6482	0,7891	0,5431	0,1971	0,5901	0,7201	0,4563	0,5939
dab_filltree2*	0,2854	0,8174	0,4092	0,2859	0,3400	0,2796	0,1865	0,1648	0,7603	0,7939	0,4323
diehard_runs*	0,5100	0,3717	0,8628	0,4501	0,8081	0,9790	0,6927	0,4991	0,1212	0,6766	0,5971
diehard_craps*	0,2298	0,8773	0,2641	0,7589	0,2988	0,5364	0,6011	0,3626	0,4948	0,3664	0,4790
Mean	0,4436	0,5787	0,5261	0,5724	0,4970	0,4272	0,4666	0,5232	0,6194	0,5574	0,5212

The second test suite, which was applied to the QRNG, was *Big Crush* from *TestU01*. This suite uses a much larger number of random data compared to *Dieharder* in order to remove suspicious p-values, i.e. too close to 0 or 1, which are due to less probable (although not impossible) data streams and not to failures of the generator under test. As shown in Table 2, our QRNG successfully passed all tests of *TestU01*.

Table 2. Results from the execution of the Big Crush test suite (TestU01). The suite was executed twice. All the tests were passed in both cases.

Test	Result	Test	Result
SerialOver	PASSED	WeightDistrib	PASSED
CollisionOver	PASSED	SumCollector	PASSED
BirthdaySpacings	PASSED	MatrixRank	PASSED
ClosePairs	PASSED	Savir2	PASSED
SimpPoker	PASSED	GCD	PASSED
CouponCollector	PASSED	RandomWalk1	PASSED
Gap	PASSED	LinearComp	PASSED
Run (sknuth)	PASSED	LempelZiv	PASSED
Permutation	PASSED	Fourier3	PASSED
CollisionPermut	PASSED	LongestHeadRun	PASSED
MaxOft	PASSED	PeriodsInStrings	PASSED
SampleProd	PASSED	HammingWeight2	PASSED
SampleMean	PASSED	HammingCorr	PASSED
SampleCorr	PASSED	HammingIndep	PASSED
AppearanceSpacings	PASSED	Run (sstrings)	PASSED

Bibliography

[1] S. Cova, M. Ghioni, A. Lacaita, C. Samori and F. Zappa. (1996). Appl. Opt. , 35 (1956).

[2] N. Gisin, G. Ribordy, W. Tittel and H. Zbinden. (2002). Rev. Mod. Phys. , 74 (145).

[3] Boyle, P. P. (1977). J. Fin. Econ. , 4 (23).

[4] G. Winkler. (2002). Image Analysis, Random Fields and Markov Chain Monte Carlo Methods: A Mathematical Introduction. Berlin: Springer.

[5] I. Marcikic, H. de Riedmatten, W. Tittel, H. Zbinden, M. Legré and N. Gisin. (2004). Phys. Rev. Lett. , 93 (180502).

[6] T. Saito, K. Ishii, I. Tatsuno, S. Sukagawa, and T. Yanagita. (2010). Randomness and Genuine Random Number Generator With Self-testing Functions. Joint International Conference on Supercomputing. Tokyo: edited by K. Todani and T. Takeda.

[7] C. R. S. Williams, J. C. Salevan, X. Li, R. Roy and T. E. Murphy. (2010). Opt. Express , 18 (23584).

[8] B. Sunar, W. J. Martin and D. R. Stinson. (2007). IEEE Trans. Com. , 56, 109-119.

[9] S. Tisa, A. Tosi and F. Zappa. (2007). Opt. Express , 15 (2873).

[10] S. Tisa, F. Guerrieri and F. Zappa. (2008). Opt. Express , 16 (2232).

[11] G. Marsaglia and A. Zaman. (1993). Comp. Math. Appl. , 26 (1).

[12] Brown, R. G. (2012). DieHarder: A Gnu Public License Random Number Tester. Duke University Physics Department Durham, NC 27708-0305.

[13] P. L'Ecuyer and R. Simard. (2007). ACM Trans. Math. Softw. , 33 (22).

[14] Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A.Heckert, J. Dray and S. Vo. (2010). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Special Publication 800-22, Revision 1. National Institute of Standards and Technology. Gaithersburg.

[15] G. Marsaglia. (1995). The marsaglia random number cdrom including the diehard battery of tests of randomness.

[16] http://www.phy.duke.edu/~rgb/General/dieharder.php

[17] M. Ghioni, A. Gulinatti, I. Rech, F. Zappa, and S. Cova. (2007). IEEE J. Select. Topics Quantum Electron., 13 (852).

[18] S. Cova, M. Ghioni, A. Lacaita, C. Samori and F. Zappa. (1996). Appl. Opt. , 35 (1956).

QRN interfaces

Front view *Back view*

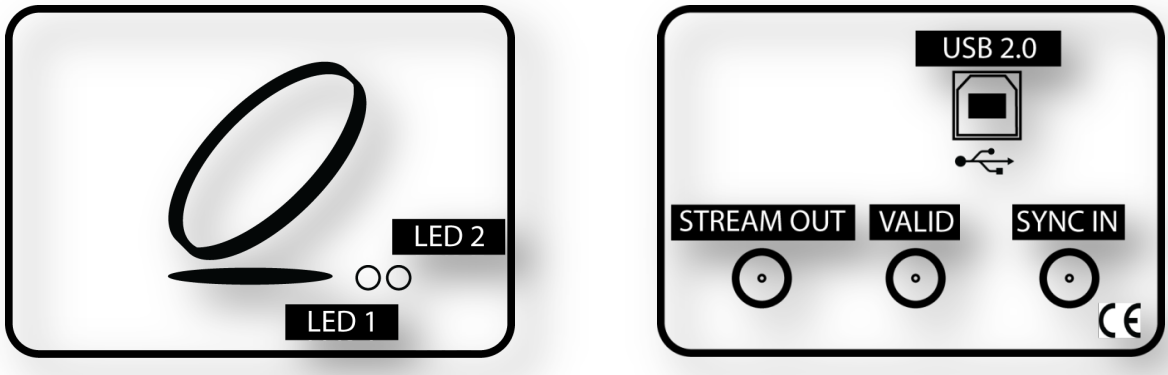


Figure 3. Quantum random number generator front and back views.

USB	High-speed USB 2.0 connector. Used both for transferring data to the PC and to power the module. The device requires less than 500 mA during operation. It is recommended to use short USB cables (< 2 m).
STREAM OUT	SMA output able to drive a 50 Ohm terminated transmission line. Electric pulses of 3.3V LVCMOS are generated at this output. Low (V_{OL}) and high (V_{OH}) voltage levels correspond to the logic values '0' and '1'. This output is not enabled when the QRN is set to "software mode".
SYNC IN	This SMA input requires a 3.3V-LVCMOS pulse (V_{IH}, 5V tolerated). The input impedance is 50 Ohm DC. This input accepts a periodic clock signal. A new random bit is generated at STREAM OUT on the rising edge of the signal. This output is not enabled when the QRN is set to "software mode".
VALID	SMA output able to drive a 50 Ohm terminated transmission line. Electric signals of 3.3V LVCMOS are generated at this output. This signal is active high. When bits from the STREAM OUT output are valid, this signal goes high. During start-up and device sleep or if an error occurs, this signal is low. This output is not enabled when the QRN is set to "software mode".

QRN operation

In order to power on the QRN just connect it to a PC or to a USB wall mount power adaptor. Once the QRN module is connected, 5 to 10 seconds are required before the internal firmware is fully functional. When the QRN is powered on, it starts always in *hardware mode*: the input and output SMA connectors are enabled and the random stream is transferred to an external hardware by means of attached cables. This means that it is possible to use the QRN without connecting it to a PC. By using the VisualQRN PC software (see the next paragraphs), or the supplied APIs, it is possible to change the working mode of the QRN to *software mode:* in this way the data are transferred to the computer and the hardware input and outputs are disabled. When in *hardware mode*, the QRN's random bits are clocked out by providing a proper signal at the SYNC IN input. The SYNC IN signal can be periodic or aperiodic but the proper timings described in paragraph "Electrical Characteristics" must be always ensured. The SYNC IN has also a maximum allowed frequency which is determined by the QRN grade and is shown also in Table 3. If a frequency higher than the maximum one is provided, the QRN will signal this ERROR condition by changing the colour of the front LEDs according to Table 4 and by stopping any generation of random bits. In order to enable again the generation of the random bits, the frequency at SYNC IN must be decreased below the maximum one, and the QRN reset by unplugging the USB cable.

It's thus up to the user to acquire data from STREAM OUT synchronously with SYNC IN. It must be also stressed that acquiring data from STREAM OUT without providing SYNC IN or not respecting proper timings could result in a not-random stream (see next paragraph). The cable length, i.e. the propagation delays, of the three connected signals MUST ALWAYS be considered, since Figure 4 **timings are measured at the QRN input/outputs with no cables attached.**

During the *hardware mode* operation even the VALID status must always be checked. As already described, when VALID's status is high the random bits are valid but when it is low they are not. As a consequence the user should not sample the STREAM OUT status whenever the VALID status is low. The following two examples will show how to properly use the VALID output. Example (1): immediately after the power up, the user must wait for the VALID status to go high, before clocking out the random bits. Example (2): as previously described, when a frequency, higher the maximum allowed one, is applied to the SYNC IN, the QRN not only stops the generation of random bits and signals this condition through the front LEDs colour, but also reduces the VALID status to the low state; in order to exit this condition the frequency must be reduced and the QRN restarted.

User must also check VALID signal before acquiring data from STREAM OUT.

The QRN module enters a *deep sleep mode* whenever it is not used for a while, independently of being in *hardware mode* or in *software mode*. If the QRN is in software mode, the exit form the *deep sleep mode* is straightforward and is correctly handled by the QRN API or by the Visual QRN. If the QRN is in *Hardware mode*, the VALID signal goes low as soon as the *deep sleep* starts. In order to get the QRN out of the *deep sleep mode*, a sync signal must be sent to the QRN. The exit from the sleep state requires some time, during which the generator is restored to operating conditions. At the end of this period, the VALID signal goes high again: of course, no random bit should be sampled at the STREAM OUT before the VALID status has reached the high level.

If no SYNC IN is provided and no error occurred, the VALID signal will remain high until the device enters sleep mode (i.e. after about 5 seconds without any rising edge of the SYNC IN).

Electrical Characteristics

Voltages and timings

The logic values and the timing properties of SYNC IN and STREAM OUT are described in Table 3. Typical propagation delays of about 14 ns from the rising edge of SYNC IN and the generation of a new random bit at STREAM OUT are expected. The VALID output has an asynchronous response to SYNC IN. In fact, it is generated by the internal control electronics. Whenever an error occurs, the SYNC IN input is disabled and the STREAM OUT is set to '0'.

Table 3. Electrical Characteristics of the QRN measured at room temperature (25°C)

Symbol	Parameter	Signal	Grade	Min.	Typ.	Max.	Unit
V_{IH}	Input logic-high voltage (5V tolerant)	SYNC IN	All	2.4		3.3	V
V_{IL}	Input logic-low voltage	SYNC IN	All	0		0.8	V
V_{OH}	Output logic-high voltage (50Ω load)	STREAM OUT, VALID	All		2.7		V
V_{OL}	Output logic-low voltage (50Ω load)	STREAM OUT, VALID	All		0		V
t_{rep}	Minimum time between two synchronization voltage pulses	SYNC IN	200Mb/s 120Mb/s 64 Mb/s 32 Mb/s 16 Mb/s		5.2 8.5 16 32 63		ns
t_{min}	Minimum duration of a synchronization voltage pulse	SYNC IN	All		4		ns
t_{PD}	Propagation delay	STREAM OUT	All	11	13.4	16	ns
t_{HL}	Minimum duration of a logic output value	STREAM OUT	200Mb/s 128Mb/s 64 Mb/s 32 Mb/s 16 Mb/s		5.2 8.5 16 32 63		ns
t_{or}, t_{of}	Output rising and falling times (10%-90%)	STREAM OUT	All		1		ns
t_{ir}, t_{if}	Input rising and falling times (10%-90%)	SYNC IN	All			0.2*t_{HL}	ns

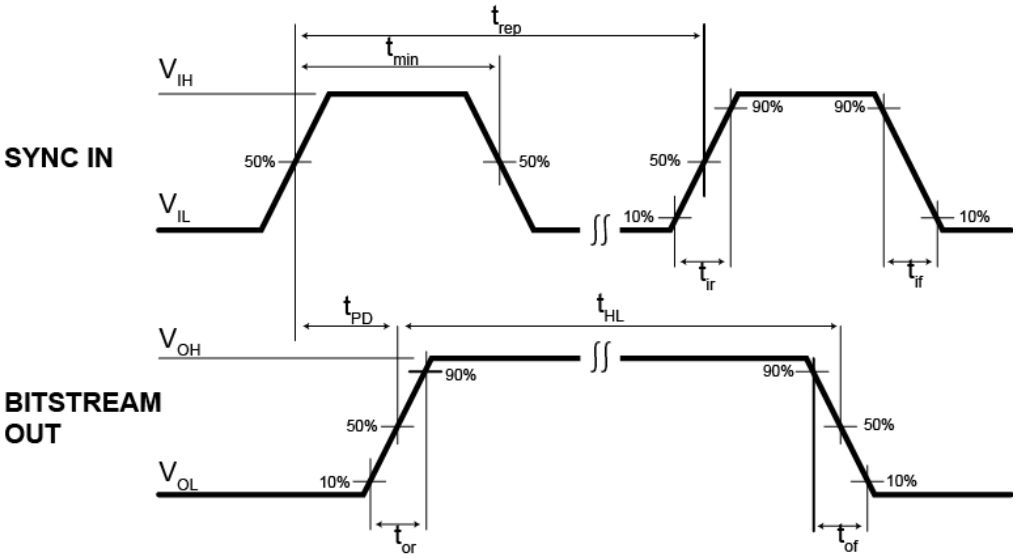


Figure 4. Timing properties of the hardware interface.

QRN status

Two LEDs on the front panel of the device provide the information on the actual state of the QRN. Table 4 explains the possible states.

Table 4. Coding for the two multi-colour LED on the front panel

LED 1	LED 2	Message
●	●	Device ON and idle. If blinking, the QRN is in "sleep mode", it will wake as soon as data are requested either by USB (if in software mode) or through SYNC IN (if in hardware mode)
●	●	Reading data
●	●	Hardware error Code #1. Contact Micro Photon Devices for assistance
●	●	USB power error: change USB port or use an externally powered USB hub
●	●	Hardware error Code #2. Contact Micro Photon Devices for assistance. In hardware mode QNN stops working. In software mode and If "security mode" is enabled, random bits are still provided but they come exclusively from the internal PRNG. To indicate this event, this LED starts blinking.
●	●	Hardware mode: the bit stream is read too fast for the device. Decrease the frequency of the voltage pulses at the SYNC IN input
●	●	Frequency test error. Too long sequence of bits equal to '0' or '1'
●	●	Hardware error Code #3. Contact Micro Photon Devices for assistance

Power consumption

QRN is fully USB powered and does not require any additional power supply unit. The amount of power depends on the device grade and is shown in Table 5. If no data are requested for longer than 5 seconds, the device enters the idle state.

Table 5. QRN power consumption as the function of the grade and state

Grade [Mb/s]	Idle [mW]	Running [mW]
16	850	1100
32	850	1100
64	850	1100
128	850	1200
200	850	1300

QRN Software interface – VisualQRN (Windows only)

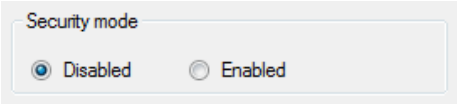
VisualQRN is a basic graphic user interface to test the quantum random number generator and to familiarize with its performance. The software includes a basic check of the random stream and a direct access to the generator parameters. As an option it might also provide a very useful and powerful feature: a server interface that allows to send the random data over the network. Due to the time needed by the internal firmware to be fully working, the VisualQRN should be started always 15 seconds after having powered the QRN. Failing to do so will generate and error by the software. Simply close and restart the software.

Installation

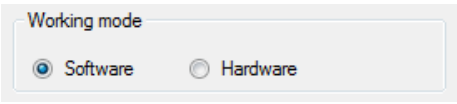
Simply double-click the desired installer (32- or 64-bit) and follow the guided installation procedure. During the installation, a command window will appear for device driver installation. Wait for a command line window to be opened and wait for the following sentence to appear: *"The driver was successfully installed"* message appears. Then press Enter to complete the installation.

Settings and operation

There are only two major settings for the device:



Enable or disable the XOR between the QRN stream and a pseudo random number generator of the KISS family. This is useful in case a complete hardware failure of the QRN occurs: random data are still provided by the KISS PRNG, enabling the user to substitute the failing device with minimum out-of-service time. This event is signalled with a clear message in VisualQRN window, and with the blinking of LED1 on the QRN.



Select the working mode for the device: (*Software*) the data are transferred to the computer via the API functions. (*Hardware*) the input and output SMA connectors are enabled. The random stream is transferred to an external hardware. Upon connection to the USB, QRN starts in hardware mode, so it possible to use it without a PC.

Three different applications are included:

1. ***Stream Analyser*** – It features a basic check of the random bit stream.
2. ***Random stream generator*** – Allows to save a random data stream on the hard disk.
3. ***QRN server (optional)*** – A basic multithreaded TCP/IP server to share random numbers over the network.

Stream Analyser

With this panel, shown in Figure 5, a basic check of the random bit stream is performed in real time. The size of the random data stream (i.e. the number of bytes used for the processing) is equal to 1MB = 1,000,000 bytes.

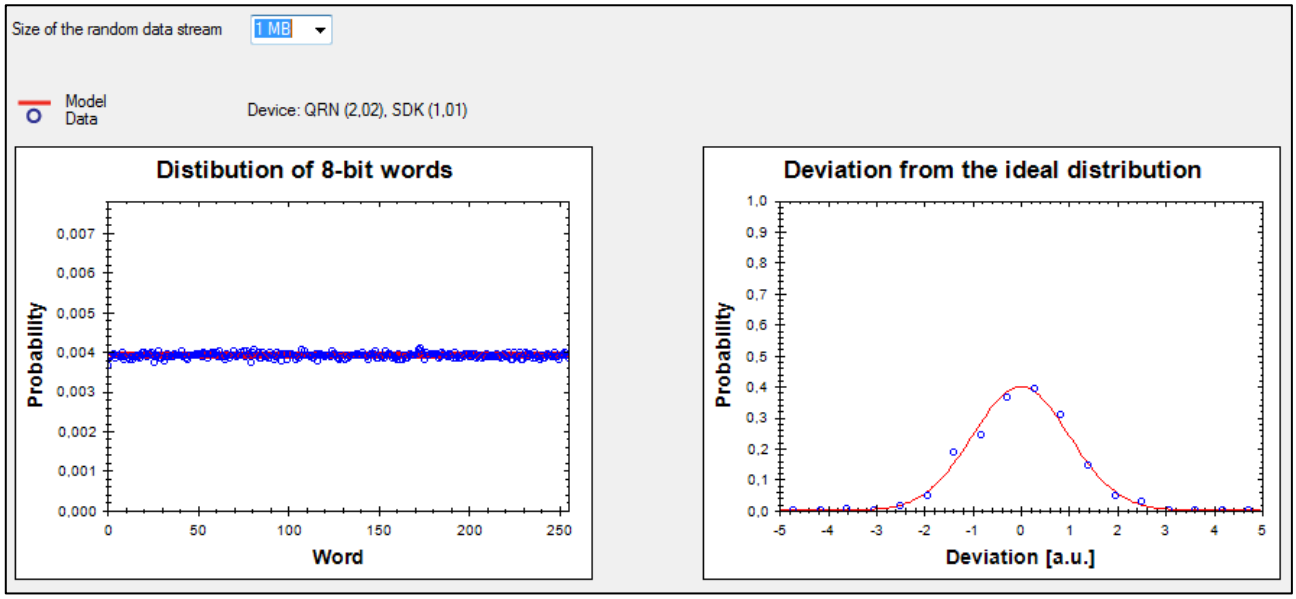


Figure 5. (Left) Frequency distribution of 8-bit words calculated from a 1 MB stream. (Right) Frequency distribution of the deviations between model and measured data. The red curves (-) are the model distributions which are expected for perfectly random data streams. The blue curves (o) are the frequency distributions obtained from a 1MB of random data acquired by the QRN.

The stream is divided in bytes which represent integer numbers between 0 and 255. If the stream is perfectly random, the occurrence frequency of each 8-bit number is the same. The frequency distribution of the 8-bit words (*F*) is constructed and it should converge in law to a discrete uniform distribution:

$$F(i) = \lim_{k\to\infty} F_k(i) = \lim_{k\to\infty} \frac{n_i}{k} = \frac{1}{256}, \qquad (1)$$

where *k* is the number of used bytes and the n_i, *i = 0 ÷ 255*, count the number of occurrences of each number *i* in the stream. The n_i are random variables because they assume different values depending on the analysed random stream. The frequency (F_k, data) and the theoretical distributions (F, model) are then plotted on the left axis of the windows form (Figure 5 - Left).

These two distributions do not match perfectly. In fact, the frequency and uniform distributions are equivalent only in the limit case $k \to \infty$. The frequency distribution will be scattered around the uniform distribution and the magnitude of this deviation depends on *k*.

A second frequency distribution of the deviation between the two distributions is then constructed. We can make two assumptions, which are valid for an ideal random stream:

1. The n_i are independent and identically distributed random variables
2. The n_i are distributed according to a Poisson distribution of mean *k/256*.

In fact, each 8 bit word has the same occurrence probability for an ideal random stream of bytes. The second assumption is derived by the properties of a Bernoulli process. The occurrence of a number *i* in a stream of *k* bytes can be seen as *k* repetitions of independent Bernoulli trials of mean *p = 1/256*. Considering that *p* is "small" and *k* is "large", i.e. p < 0.01 and *k > 100*, the occurrence of a number *i* in the stream is distributed according to a Poisson law with mean k*p = k/256. This property is usually named "law of rare events".

When the product *k*p* becomes very large, i.e. *k*p* > 100, the Poisson distribution converges in probability to a discrete Gauss distribution of mean and variance equal to $\lambda = k/256$ according to the *central limit theorem*. The distribution of each random variable n_i *is* thus the following:

$$G(n_i) = \frac{1}{\sqrt{2\pi\lambda}}\, e^{-\frac{(n_i-\lambda)^2}{2\lambda}} = \frac{1}{\sqrt{\frac{2\pi k}{256}}}\, e^{-\frac{\left(n_i-\frac{k}{256}\right)^2}{2*\frac{k}{256}}} \tag{2}$$

We define now a set of random variables h_i, which represents the normalized deviation of n_i from the model distribution.

$$h_i = \left(n_i - \frac{k}{256}\right) / \sqrt{\frac{k}{256}} \tag{3}$$

After a variable change in eq. (2), one can easily show that the h_i follow a *standard Gauss distribution* independently on *k*.

$$G(h_i) = \frac{1}{\sqrt{2\pi}}\, e^{-\frac{(h_i)^2}{2}} \tag{4}$$

This distribution is the same for each 8-bit number. We can therefore extract 256 values which are distributed according to eq. (4) from the stream. These values are calculated as follows.

1. The n_i are calculated from a stream of *k* bytes, e.g. how often the number 25 appears in the stream
2. Eq. (3) is applied to each n_i to calculate the h_i
3. A frequency distribution of the h_i is then constructed to inspect visually the match between eq. (4) and the data obtained from the stream (Figure 5 - Right).

In conclusion, a random stream should not show any pattern on both axis and the constructed frequency distributions should match the limit one.

Random stream generator

With this panel it is possible to save a random data stream on the hard disk. By selecting first the desired amount of megabytes and then pressing the "Run" button, the stream is saved on the hard disk in binary format. Alternatively, the generation rate can be tested (Please note that MPD use the following unit of measure : 1MB = 1,000,000 bytes).

WARNING: The generation rate can be accurately measured only for large data streams due to the automatic data buffering in the computer RAM. Measured generation rate will always be a little lower than the QRN grade, due to the small amount of time needed by the PC to send the commands to the QRN).

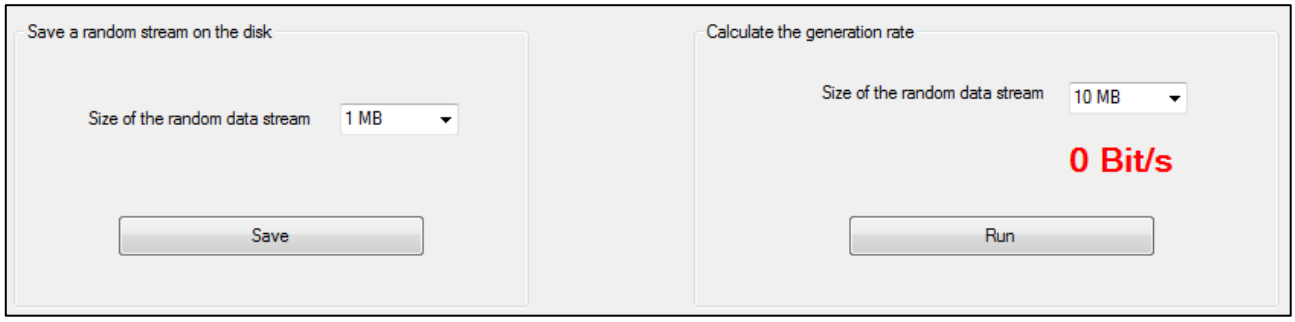


Figure 6. (Left) Generate and save a random data stream on the hard disk. (Right) Calculate the random bit generation rate for a target device.

Some of the most popular battery of tests for random number generators can be then applied to the obtained streams. These test suites have been created to recognize deviations of random data streams from *true* randomness. Some statistical test suites for random numbers are:

1. Test U01 http://www.iro.umontreal.ca/~simardr/testu01/tu01.html
2. NIST http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html

3. Diehard http://stat.fsu.edu/pub/diehard/

4. Dieharder http://www.phy.duke.edu/~rgb/General/dieharder.php

The Micro Photon Devices QRN passed all the above listed battery of tests.

QRN server (Optional)

With this panel, shown in Figure 7, it is possible to start a basic multithreaded TCP/IP server to share random numbers over the network. Please contact *sales@micro-photon-devices.com* for pricing and availability.

This simple server application can be used to provide random numbers to several computers connected to a network. TCP/IP is a standard communication protocol which is supported by most programming languages. A fast way to test the QRN from different computers or operating systems is to run *QRN Server* on a windows machine and to download on the local computers the random data streams.

Port: Select a free port on the host computer

Start/Stop: Allow/Stop connections to the server

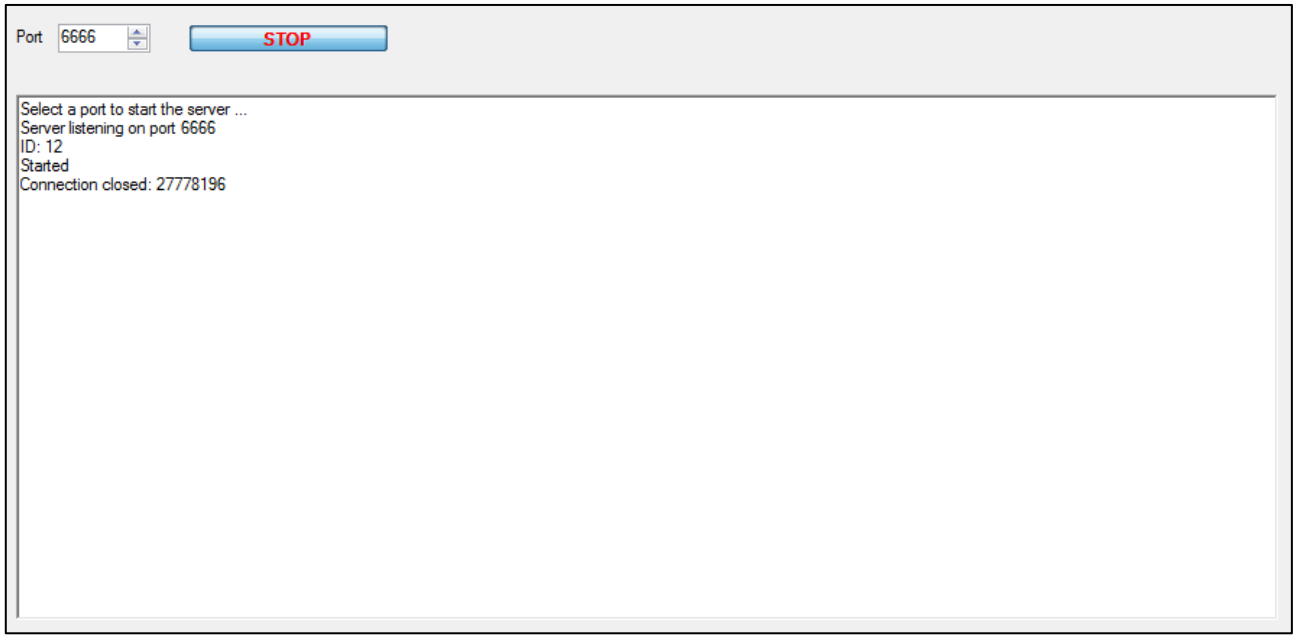


Figure 7. Simple server interface to share the random data stream over the network.

QRN SDK

QRN is provided with a Software Development Kit (SDK), composed by a shared library and few example projects. The SDK allows the user to incorporate random numbers from QRN in their own application, and it is available for Windows, Mac OS X and Linux operating systems, both in 32- and 64-bit versions.

Please read the related documentation for more information about installing and using the SDK.

System requirements

- High-speed USB 2.0 interface
- Host computer (minimum requirements)
    - 300 MHz processor and 256 MB of RAM
- Supported operating systems
    - VisualQRN
        - Microsoft Windows XP, Vista, 7, 8, 32 or 64 bit versions
    - SDK
        - Microsoft Windows XP, Vista, 7, 8, 32 or 64 bit versions
        - Linux Ubuntu 12.04 LTS or compatible distributions, 32 or 64 bit versions. Different distributions should work, but were not tested.
        - Mac OS X 10.7.5 and above

QRN mechanical dimensions

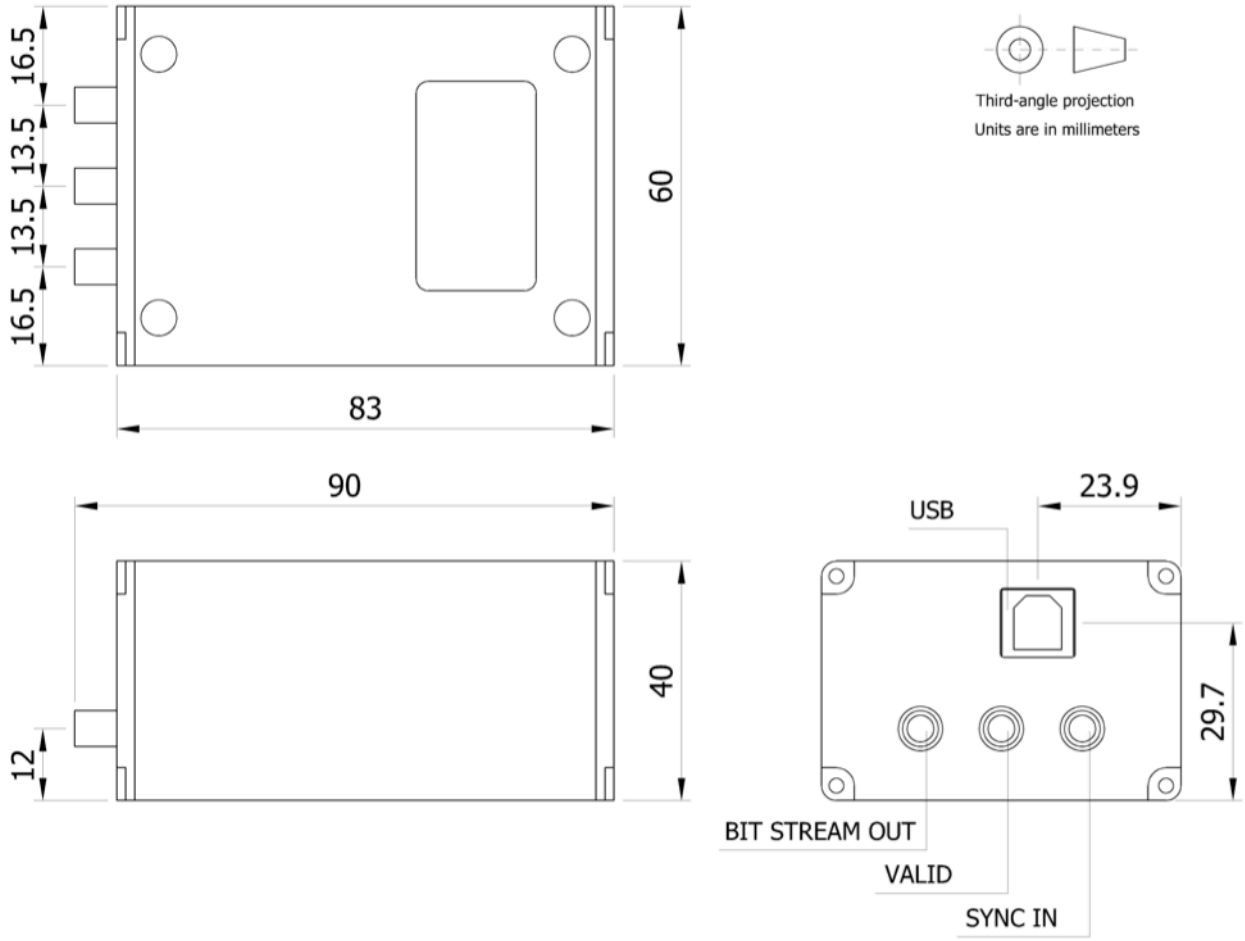


Figure 8. Dimension of the case of the QRNG. The values are in millimetres.

Copyright and disclaimer